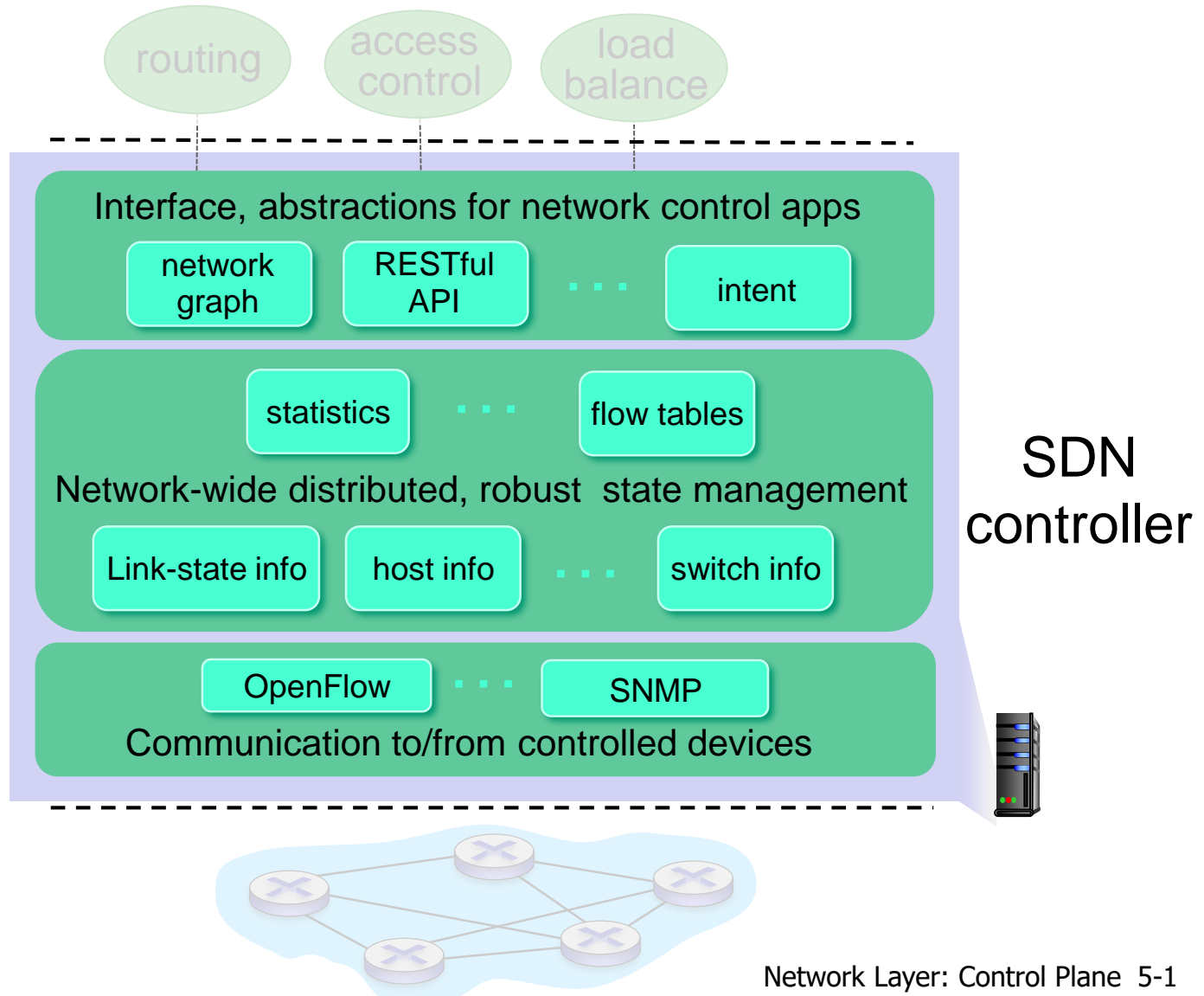


# Components of SDN controller

**Interface layer to network control apps:** abstractions API

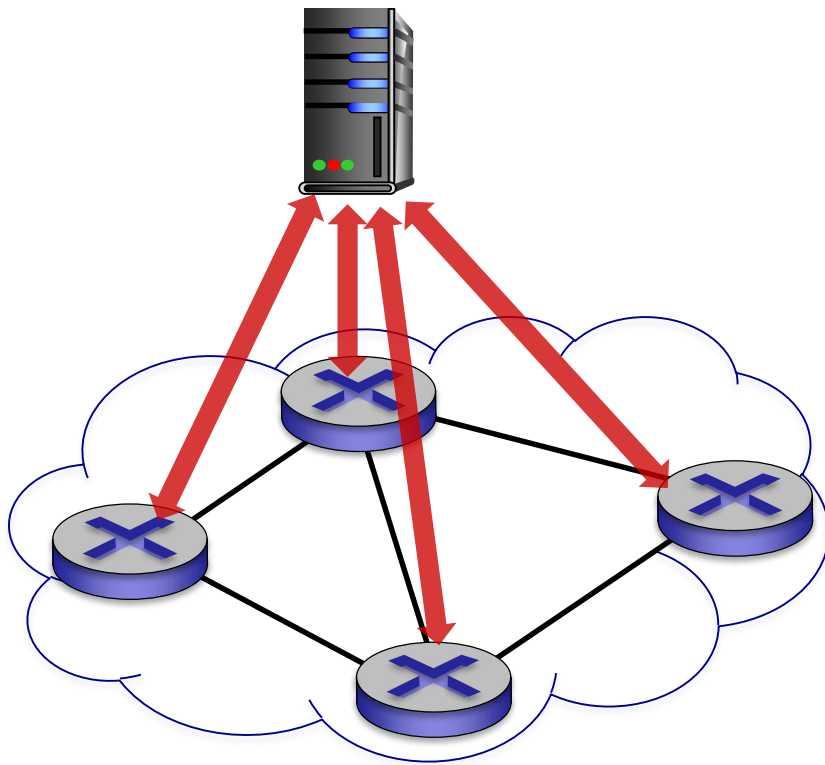
**Network-wide state management layer:** state of networks links, switches, services: a *distributed database*

**communication layer:** communicate between SDN controller and controlled switches



# OpenFlow protocol

OpenFlow Controller

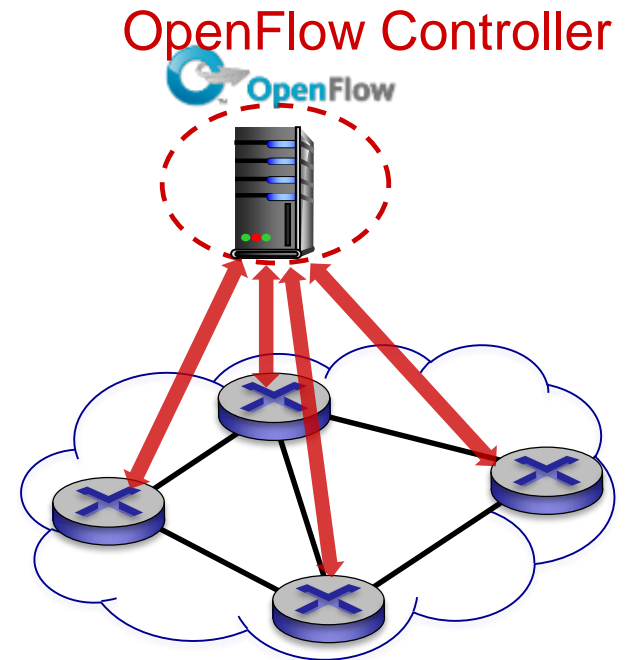


- operates between controller, switch
- TCP used to exchange messages
  - optional encryption
- three classes of OpenFlow messages:
  - controller-to-switch
  - asynchronous (switch to controller)
  - symmetric (misc)

# OpenFlow: controller-to-switch messages

## Key controller-to-switch messages

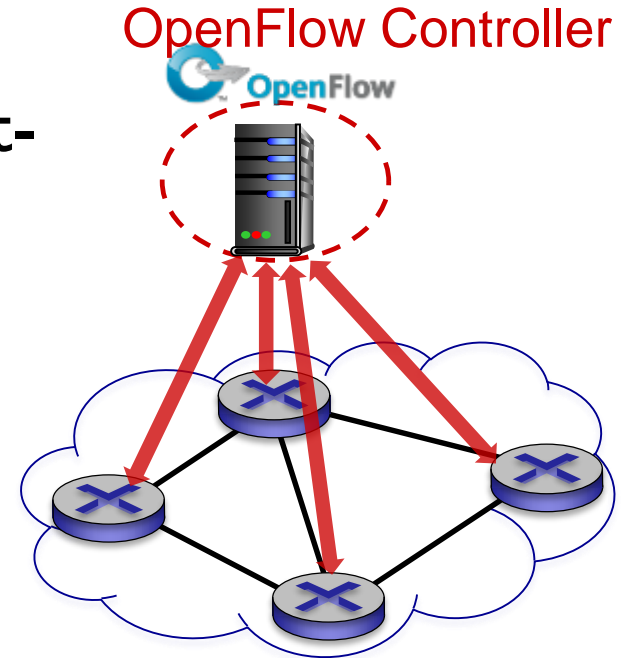
- **features:** controller queries switch features, switch replies
- **configure:** controller queries/sets switch configuration parameters
- **modify-state:** add, delete, modify flow entries in the OpenFlow tables
- **packet-out:** controller can send this packet out of specific switch port



# OpenFlow: switch-to-controller messages

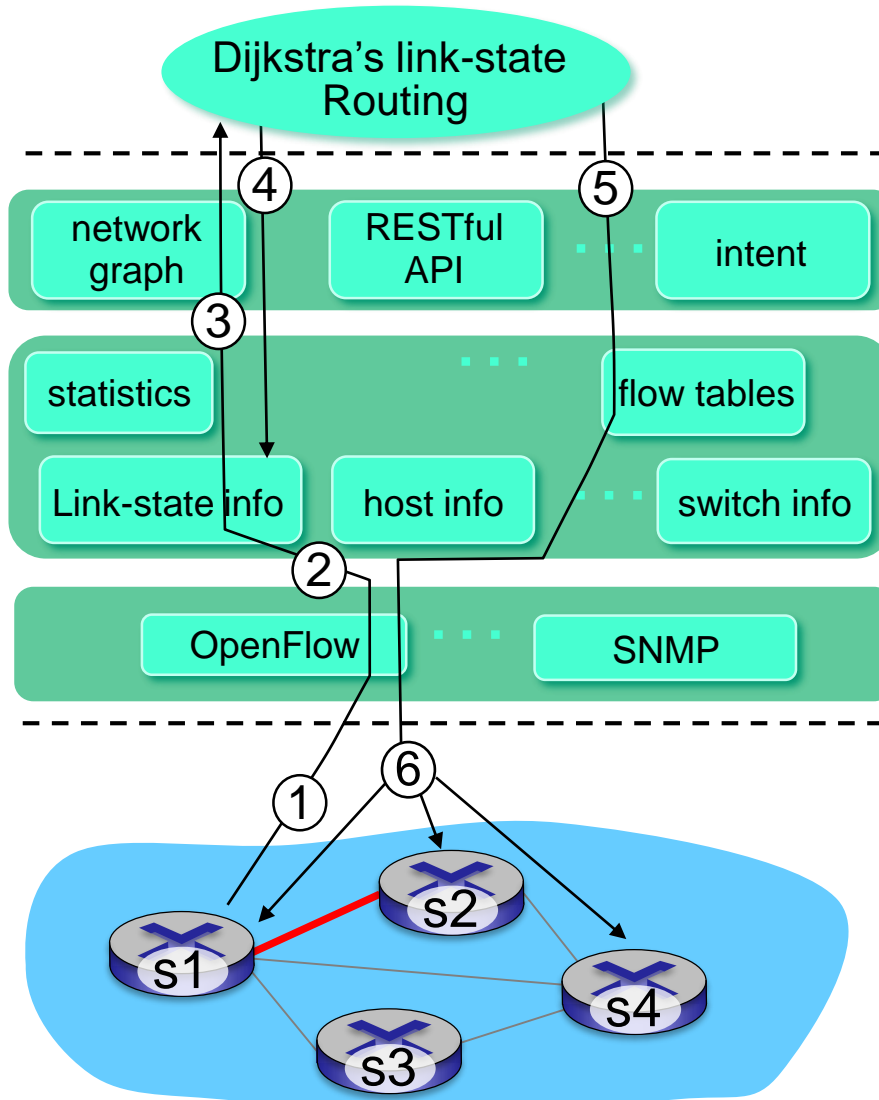
## Key switch-to-controller messages

- **packet-in:** transfer packet (and its control) to controller. See packet-out message from controller
- **flow-removed:** flow table entry deleted at switch
- **port status:** inform controller of a change on a port.



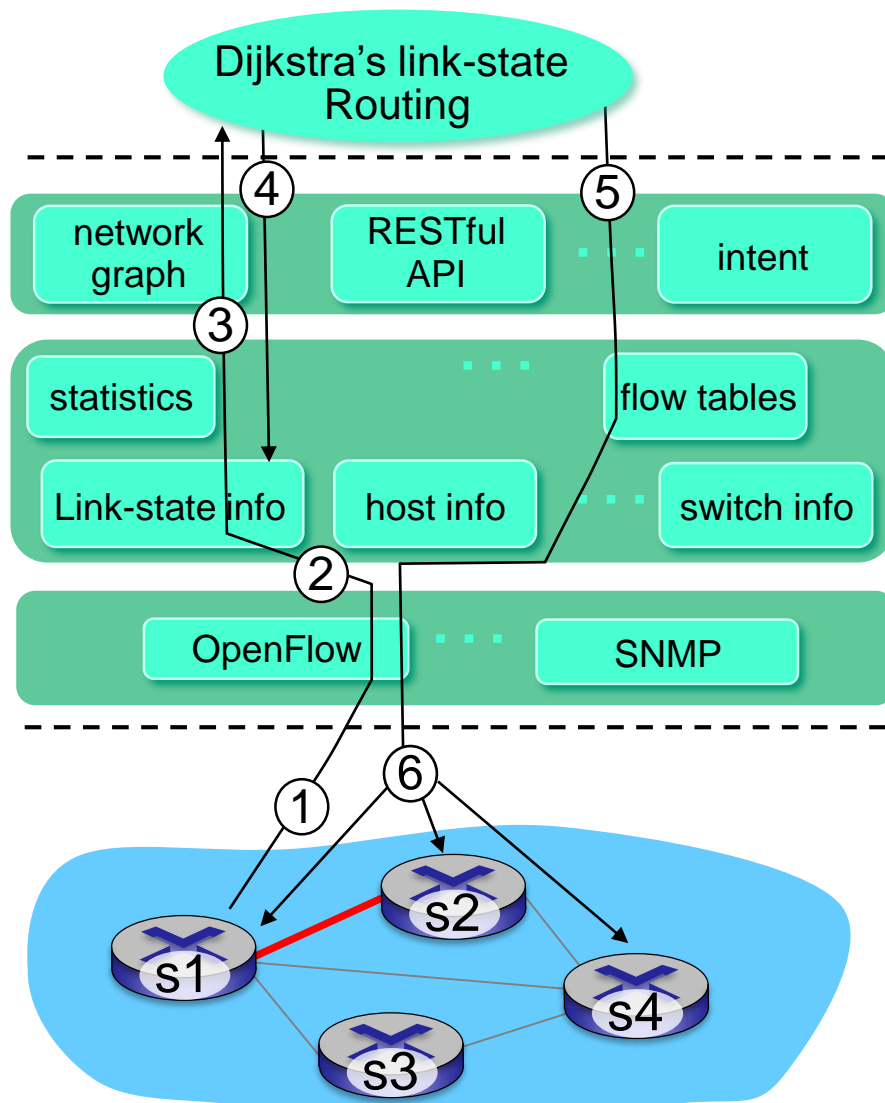
Fortunately, network operators don't “program” switches by creating/sending OpenFlow messages directly. Instead use higher-level abstraction at controller

# SDN: control/data plane interaction example



- ① SI, experiencing link failure using OpenFlow port status message to notify controller
- ② SDN controller receives OpenFlow message, updates link status info
- ③ Dijkstra's routing algorithm application has previously registered to be called when ever link status changes. It is called.
- ④ Dijkstra's routing algorithm access network graph info, link state info in controller, computes new routes

# SDN: control/data plane interaction example



- ⑤ link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed
- ⑥ Controller uses OpenFlow to install new tables in switches that need updating

# Chapter 5: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector

5.3 intra-AS routing in the Internet: OSPF

5.4 routing among the ISPs: BGP

5.5 The SDN control plane

5.6 ICMP: The Internet Control Message Protocol

5.7 Network management and SNMP

# ICMP: internet control message protocol

- used by hosts & routers to communicate network-level information

- error reporting:  
unreachable host, network, port, protocol
- echo request/reply (used by ping)

- network-layer “above” IP:

- ICMP msgs carried in IP datagrams

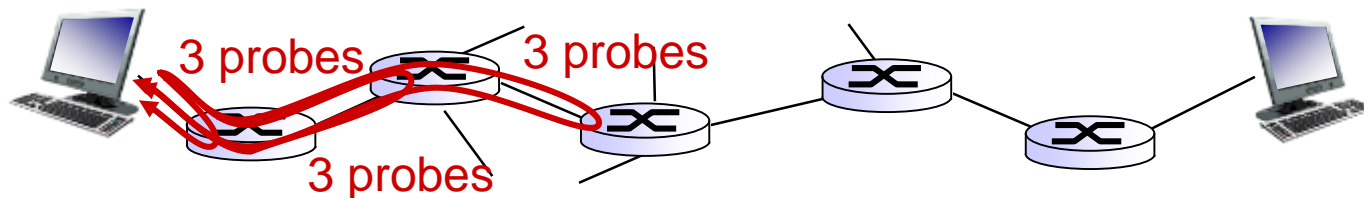
- **ICMP message:** type, code plus first 8 bytes of IP datagram causing error

<u>Type</u>	<u>Code</u>	<u>description</u>
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header



# Traceroute and ICMP

- source sends series of UDP segments to destination
    - first set has TTL = 1
    - second set has TTL=2, etc.
    - unlikely port number
  - when datagram in  $n$ th set arrives to  $n$ th router:
    - router discards datagram and sends source ICMP message (type 11, code 0)
    - ICMP message include name of router & IP address
  - when ICMP message arrives, source records RTTs
- stopping criteria:*
- UDP segment eventually arrives at destination host
  - destination returns ICMP “port unreachable” message (type 3, code 3)
  - source stops



# Link layer, LANs: outline

6.1 introduction, services

6.2 error detection,  
correction

6.3 multiple access  
protocols

6.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANs

6.5 link virtualization:  
MPLS

6.6 data center  
networking

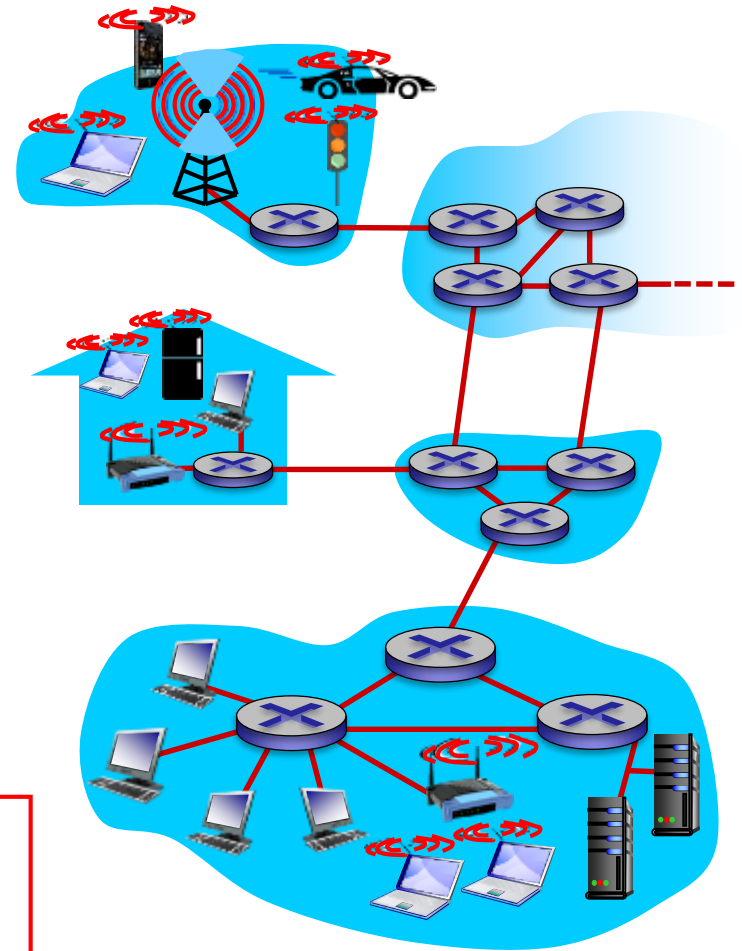
6.7 a day in the life of a  
web request

# Link layer: introduction

## *terminology:*

- hosts and routers: **nodes**
- communication channels that connect adjacent nodes along communication path: **links**
  - wired links
  - wireless links
  - LANs
- layer-2 packet: **frame**, encapsulates datagram

*data-link layer* has responsibility of transferring datagram from one node to *physically adjacent* node over a link



# Link layer: context

- datagram transferred by different link protocols over different links:
  - e.g., Ethernet on first link, frame relay on intermediate links, 802.11 on last link
- each link protocol provides different services
  - e.g., may or may not provide reliable data transfer over link

# Link layer services

## ■ *framing, link access:*

- encapsulate datagram into frame, adding header, trailer
- channel access if shared medium
- “MAC” addresses used in frame headers to identify source, destination
  - different from IP address!

## ■ *reliable delivery between adjacent nodes*

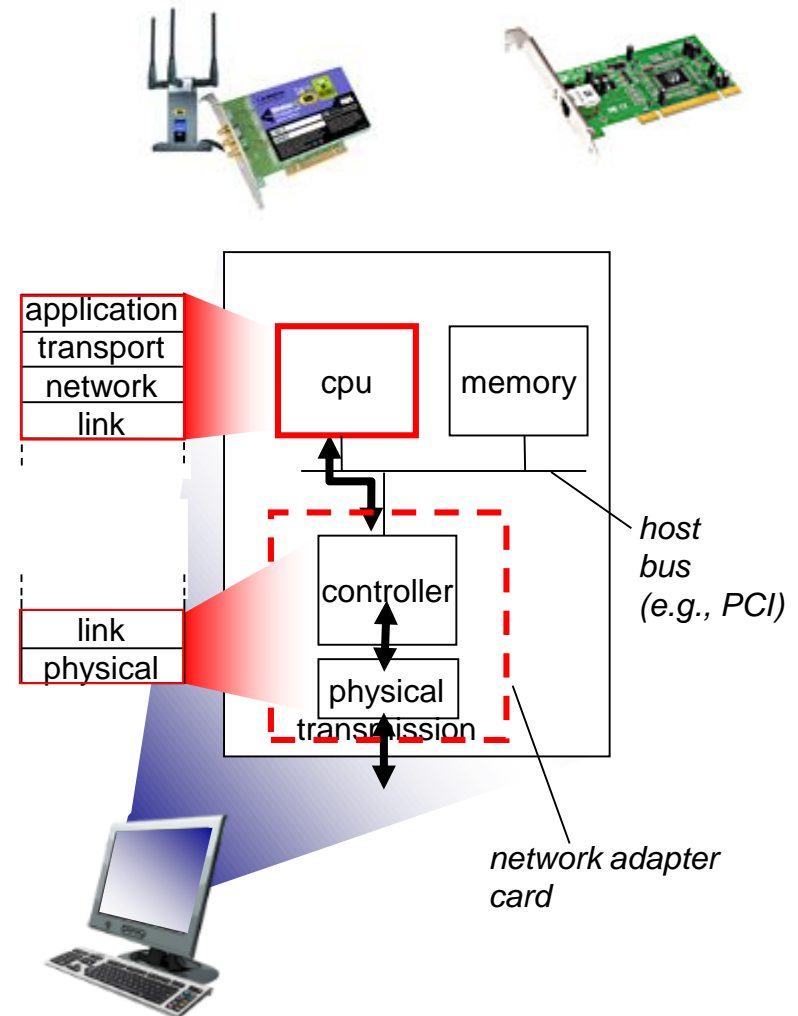
- we learned how to do this already (chapter 3)!
- seldom used on low bit-error link (fiber, some twisted pair)
- wireless links: high error rates

# Link layer services (more)

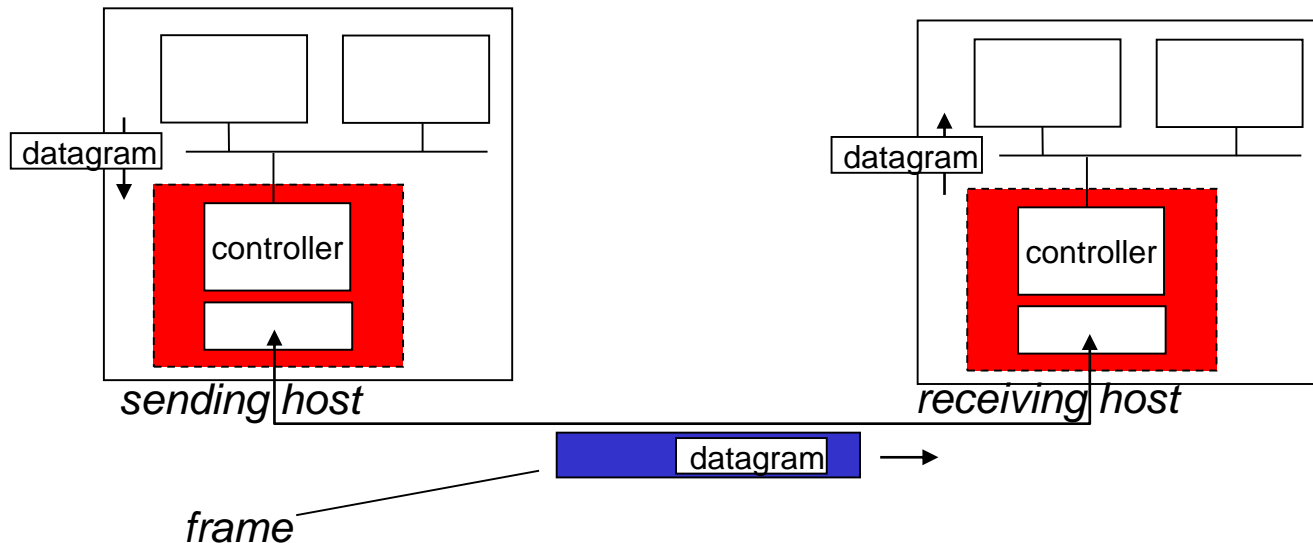
- *flow control:*
  - pacing between adjacent sending and receiving nodes
- *error detection:*
  - errors caused by signal attenuation, noise.
  - receiver detects presence of errors:
    - signals sender for retransmission or drops frame
- *error correction:*
  - receiver identifies *and corrects* bit error(s) without resorting to retransmission
- *half-duplex and full-duplex*
  - with half duplex, nodes at both ends of link can transmit, but not at same time

# Where is the link layer implemented?

- in each and every host
- link layer implemented in “adaptor” (aka *network interface card* NIC) or on a chip
  - Ethernet card, 802.11 card; Ethernet chipset
  - implements link, physical layer
- attaches into host's system buses
- combination of hardware, software, firmware



# Adaptors communicating



- sending side:
  - encapsulates datagram in frame
  - adds error checking bits, rdt, flow control, etc.
- receiving side
  - looks for errors, rdt, flow control, etc.
  - extracts datagram, passes to upper layer at receiving side



# Link layer, LANs: outline

6.1 introduction, services

6.2 error detection,  
correction

6.3 multiple access  
protocols

6.4 LANs

- addressing, ARP
- Ethernet
- switches
- VLANs

6.5 link virtualization:  
MPLS

6.6 data center  
networking

6.7 a day in the life of a  
web request

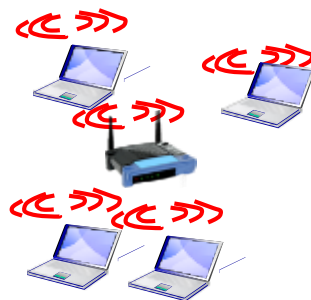
# Multiple access links, protocols

two types of “links”:

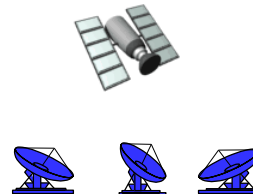
- point-to-point
  - PPP for dial-up access
  - point-to-point link between Ethernet switch, host
- *broadcast (shared wire or medium)*
  - old-fashioned Ethernet
  - upstream HFC
  - 802.11 wireless LAN



shared wire (e.g.,  
cabled Ethernet)



shared RF  
(e.g., 802.11 WiFi)



shared RF  
(satellite)



humans at a  
cocktail party  
(shared air, acoustical)

# Multiple access protocols

- single shared broadcast channel
- two or more simultaneous transmissions by nodes:  
interference
  - *collision* if node receives two or more signals at the same time

## *multiple access protocol*

- distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit
- communication about channel sharing must use channel itself!
  - no out-of-band channel for coordination

# An ideal multiple access protocol

*given:* broadcast channel of rate  $R$  bps

*Desired properties:*

1. when one node wants to transmit, it can send at rate  $R$ .
2. when  $M$  nodes want to transmit, each can send at average rate  $R/M$
3. fully decentralized:
  - no special node to coordinate transmissions
  - no synchronization of clocks, slots
4. simple

# MAC protocols: taxonomy

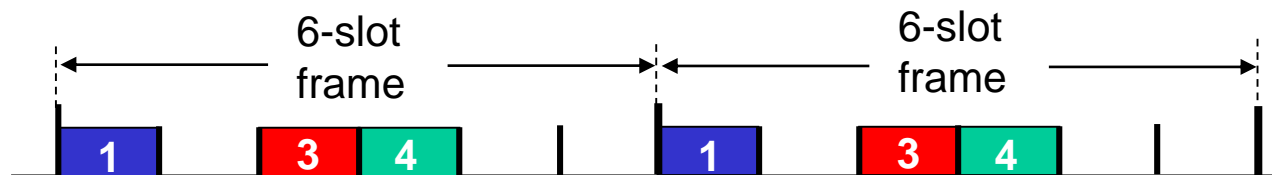
three broad classes:

- *channel partitioning*
  - divide channel into smaller “pieces” (time slots, frequency, code)
  - allocate piece to node for exclusive use
- *random access*
  - channel not divided, allow collisions
  - “recover” from collisions
- *“taking turns”*
  - nodes take turns, but nodes with more to send can take longer turns

# Channel partitioning MAC protocols: TDMA

## TDMA: time division multiple access

- access to channel in "rounds"
- each station gets fixed length slot (length = packet transmission time) in each round
- unused slots go idle
- example: 6-station LAN, 1,3,4 have packets to send, slots 2,5,6 idle



# Channel partitioning MAC protocols: FDMA

## FDMA: frequency division multiple access

- channel spectrum divided into frequency bands
- each station assigned fixed frequency band
- unused transmission time in frequency bands go idle
- example: 6-station LAN, 1,3,4 have packet to send, frequency bands 2,5,6 idle

